REMARKS

I. Introduction

In response to the Office Action dated February 8, 2007, claims 1, 3-7, 9-13 and 15-18 have been amended. Claims 1-18 remain in the application. Re-examination and re-consideration of the application, as amended, is requested.

II. Statutory Subject Matter Rejections

In paragraphs (4)-(6) of the Office Action, claims 1-18 were rejected under 35 U.S.C. §101 as being directed to non-statutory subject matter.

Applicants' attorney has amended the claims to overcome these rejections.

However, should issues still remain in this regard, Applicants' attorney requests that the Examiner indicate how the rejection can be overcome, in accordance with the directives of the Interim Guidelines for Examination of Patent Applications for Patent Subject Matter Eligibility (Guidelines) II. See also M.P.E.P. §2106. Specifically, should it be necessary, the Applicants' attorney requests that the Examiner identify features of the invention that would render the claimed subject matter statutory if recited in the claim. See Guidelines IV.B. See also M.P.E.P. §2106.

III. Prior Art Rejections

A. The Office Action Rejections

In paragraphs (7)-(8) of the Office Action, claims 1-18 were rejected under 35 U.S.C. §103(a) as being unpatentable over U.S. Patent No. 5,948,113 (Johnson) in view of U.S. Patent No. 6,785,848 (Glerum).

Applicants' attorney respectfully traverses these rejections.

B. The Applicants' Independent Claims

Independent claims 1, 7 and 13 are generally directed to providing contextual diagnostic data at a point of failure of a software program. Claim 1, which is representative, is a method for providing contextual diagnostic data at a point of failure of a software program, comprising:

(a) registering callbacks for one or more modules and sub-applications within the program;

(b) examining a call stack for the program upon failure of the program;

(c) notifying the registered callbacks for the modules and sub-applications based on the examined call stack;

(d) performing callback processing, wherein the notified callbacks of the modules and sub-applications extract and supply the contextual diagnostic data;

(e) packaging the contextual diagnostic data supplied by the notified callbacks of the modules and sub-applications; and

(f) using the packaged contextual diagnostic data for further analysis in order to troubleshoot the point of failure of the software program.

C. The Johnson Reference

Johnson describes centrally handling a runtime error or exception of a program using a central object stack and exception handling code centrally maintained within a global object manager. The global object manager is a data structure separate from the program's call stack. When a modified TRY statement is executed, a location is marked on the central object stack. During execution of a section of code after the modified TRY statement, if a new object is needed, the global object manager efficiently allocates the new object. The global object manager either allocates the new object directly from memory or attempts to re-use a previously allocated object in a cache of available objects as the new object. The new object is then registered on the central object stack and a pointer to the new object is registered on the program's call stack. This keeps the new object and associated exception handling code off the program's call stack. When an exception is thrown, the global object manager cleans up and unregisters an object which was registered on the central object stack since the marked location. If a re-use condition is met, the object is kept in the cache as an available object already allocated from memory. However, if the re-use condition is not met, the object is de-allocated from memory.

D. The Glerum Reference

Glerum describes a method for categorizing information regarding a failure in an application program module. The failure may be a crash, a set-up failure or an assert. For a crash, a name of an executable module where the crash occurred in the application program module, a version number of the executable module, a name of a module containing an instruction causing the crash, a version number of the module and an offset into the module with the crashing instruction are determined. This bucket information is then transmitted to a repository for storage in a database. The database may be examined to determine fixes for the bug that caused the crash.

E. The Applicants' Invention is Patentable Over the References

The Applicants' invention, as recited in independent claims 1, 7 and 13, is patentable over the combination of Johnson and Glerum references, because it contains limitations not taught by the combination of references. Specifically, the combination of Johnson and Glerum references does not teach or suggest the specific steps or functions recited in Applicants' claims for providing contextual diagnostic data at a point of failure of a software program.

Nonetheless, the Office Action asserts that the combination of Johnson and Glerum teaches all the limitations of Applicants' claims.

Applicants' attorney disagrees with this analysis.

With regard to the limitations "registering callbacks for one or more modules and sub-applications within the program," "examining a call stack for the program upon failure of the program," and "notifying the registered callbacks for the modules and sub-applications based on the examined call stack," the cited locations in Johnson are set forth below:

> Johnson: col. 9, lines 26-45
>
> To implement exception handling, the GOM 200 maintains a central object stack 220 of registered objects currently in use, preferably on a per-thread basis. The GOM 200 places or registers allocated objects on the thread's central object stack 220. Prior to entering the section of code where an exception may occur, the current location (stack address) on the central object stack 220 is marked and stored. If an exception occurs within the section of code, the GOM 200 handles the exception using the central object stack 220 and the centrally maintained exception handling code 210.
>
> Essentially, the exception handling code 210 within the GOM 200 pops off objects 225a-c on the central object stack 220 added or registered since the stored location or address. In this manner, the objects popped off the central object stack 220 are deemed to be unregistered from the central object stack 220. The memory (e.g., the system memory 22) associated with each of the objects are cleaned up and the object is either saved for re-use or de-allocated from system memory 22 before continuing with execution of the section of code.

The above portions of Johnson refer to a central object stack of registered objects, but the central object stack is not a call stack of the program. Indeed, the Johnson reference notes that, traditionally, objects are allocated on an application's call stack as they are needed when the program code of the application is executed, but that the central object stack is independent and separate from the application's call stack. The central object stack of Johnson is merely used to perform clean-up of objects in use when an exception occurs.

However, Johnson says nothing about registering callbacks for modules and sub-applications within the program, examining a call stack for the program upon failure of the program, and then notifying the registered callbacks based on the examined call stack. Instead, the Johnson reference is merely concerned with exception related cleanup of allocated objects in traditional try-catch blocks, using a central object stack separate from the call stack

With regard to the limitations "performing callback processing, wherein the notified callbacks of the modules and sub-applications extract and supply the contextual diagnostic data" and "packaging the contextual diagnostic data supplied by the notified callbacks of the modules and sub-applications," the cited locations in Glerum are set forth below:

> Glerum: col. 5, lines 47-67
>
> The system 200 also comprises an exception filter 220. Exception filters are well-known in the art and may be registered by program modules when the operating system 35 is started. When a failure (an exception) occurs, the exception filter 220 code is executed. For example, suppose a failure occurs while executable program 210 is executing instructions running module 215 at location 225. If executable program 210 has registered exception filter 220 with the operating system, then the exception filter 220 is executed when executable program 210 encounters an exception.
>
> In the system 200, exception filter 220 executes a failure reporting executable 230. The failure reporting executable 230 is an executable program comprising all of the instructions needed to communicate between the application program module 205 and a repository 235. The communications between the failure reporting executable 230, the application program module 205 and the repository 235 are illustrated as arrows in FIG. 2. The failure reporting executable 230 is preferably separate from the application program module 205 because of the possible instability of the application program module (having experienced a failure).

> Glerum: col. 6, lines 32-47
>
> Based upon the type of failure, the failure reporting executable 230 then determines what relevant information to retrieve from the application program module to uniquely identify, i.e. categorize, the failure. In many cases, uniquely identifying the failure means determining the location of the failure. Typically, the categorization, or location information, of the failure is sent to the repository as a bucket. A bucket is a set of information uniquely defining the location of the failure. If a bucket from one failure matches a bucket from another failure, then it is assumed that both failures are caused by the same bug. Although not always accurate (because more than one bug may be at the same location), this assumption that failures with the same bucket information are caused by the same bug allows for effective organization in the repository, as will be further described below.

The above portions of Glerum merely describe an exception filter, which is for the type of exception being handled. The exception filter then executes a failure reporting executable that,

based upon the type of failure, determines what relevant information to retrieve from the application program module to identify or categorize the failure.

However, Glerum says nothing about performing callback processing, wherein the notified callbacks of the modules and sub-applications extract and supply the contextual diagnostic data, in the context where the callbacks are registered for modules and sub-applications within the program, the call stack for the program is examined upon failure of the program, and the registered callbacks for the modules and sub-applications are notified based on the examined call stack. Instead, the Glerum reference merely describes typical exception handling by exception type, along with minidumps, bucketing and typical error reporting, while Applicants' invention is directed to providing contextual diagnostic information.

Thus, the combination of Johnson and Glerum does not render obvious Applicants' claimed invention. Moreover, the various elements of Applicants' claimed invention together provide operational advantages over the combination of Johnson and Glerum. In addition, Applicants' claimed invention solves problems not recognized by the combination of Johnson and Glerum.

The motivation for Applicants' claimed invention results from the experience in trying to find the root cause of a problem while processing customer error reports (CERs). The problem is that, when a minidump or stack dump is received by the vendor, it may not have all the context data necessary for analyzing and solving the problem.

The approach of Applicants' claimed invention is to have modules and sub-applications register "callbacks" that are called when the call stack has addresses that fall in a registered range for those modules and sub-applications. At that point, the specific callback can glean specific context data that can be used as diagnostic data. The point is that, at the time of crash, the client application that has crashed examines the stack and, based on the addresses on the stack, notifies a specific callback to extract and supply context data as diagnostic data.

Thus, Applicants' attorney submits that independent claims 1, 7 and 13 are allowable over the combination of Johnson and Glerum. Further, dependent claims 2-6, 8-12 and 14-18 are submitted to be allowable over the combination of Johnson and Glerum in the same manner, because they are dependent on independent claims 1, 7 and 13, respectively, and thus contain all the limitations of the independent claims. In addition, dependent claims 2-6, 8-12 and 14-18 recite additional novel elements not shown by the combination of Johnson and Glerum.

IV. Conclusion

In view of the above, it is submitted that this application is now in good order for allowance and such allowance is respectfully solicited. Should the Examiner believe minor matters still remain that can be resolved in a telephone interview, the Examiner is urged to call Applicants' undersigned attorney.

Respectfully submitted,

GATES & COOPER LLP
Attorneys for Applicants

Howard Hughes Center
6701 Center Drive West, Suite 1050
Los Angeles, California 90045
(310) 641-8797

Date: May 8, 2007

By: _____
Name: George H. Gates
Reg. No.: 33,500

GHG/

G&C 30566.315-US-01